

AD-A152 255

PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLS
(U) HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB
J H REIF ET AL. MAR 84 TR-85-84 N00014-80-C-0647

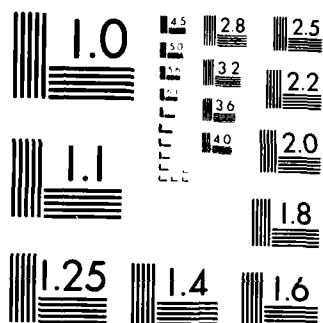
141

UNCLASSIFIED

F/G 9/2

NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A152 255

②

PARALLEL TIME $O(\log N)$ ACCEPTANCE
OF DETERMINISTIC CFLs

Philip N. Klein
John Reif

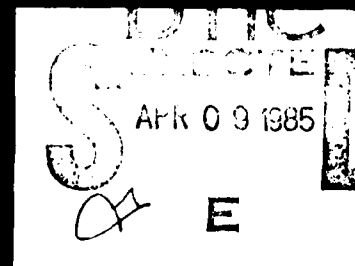
TR-05-84

Harvard University

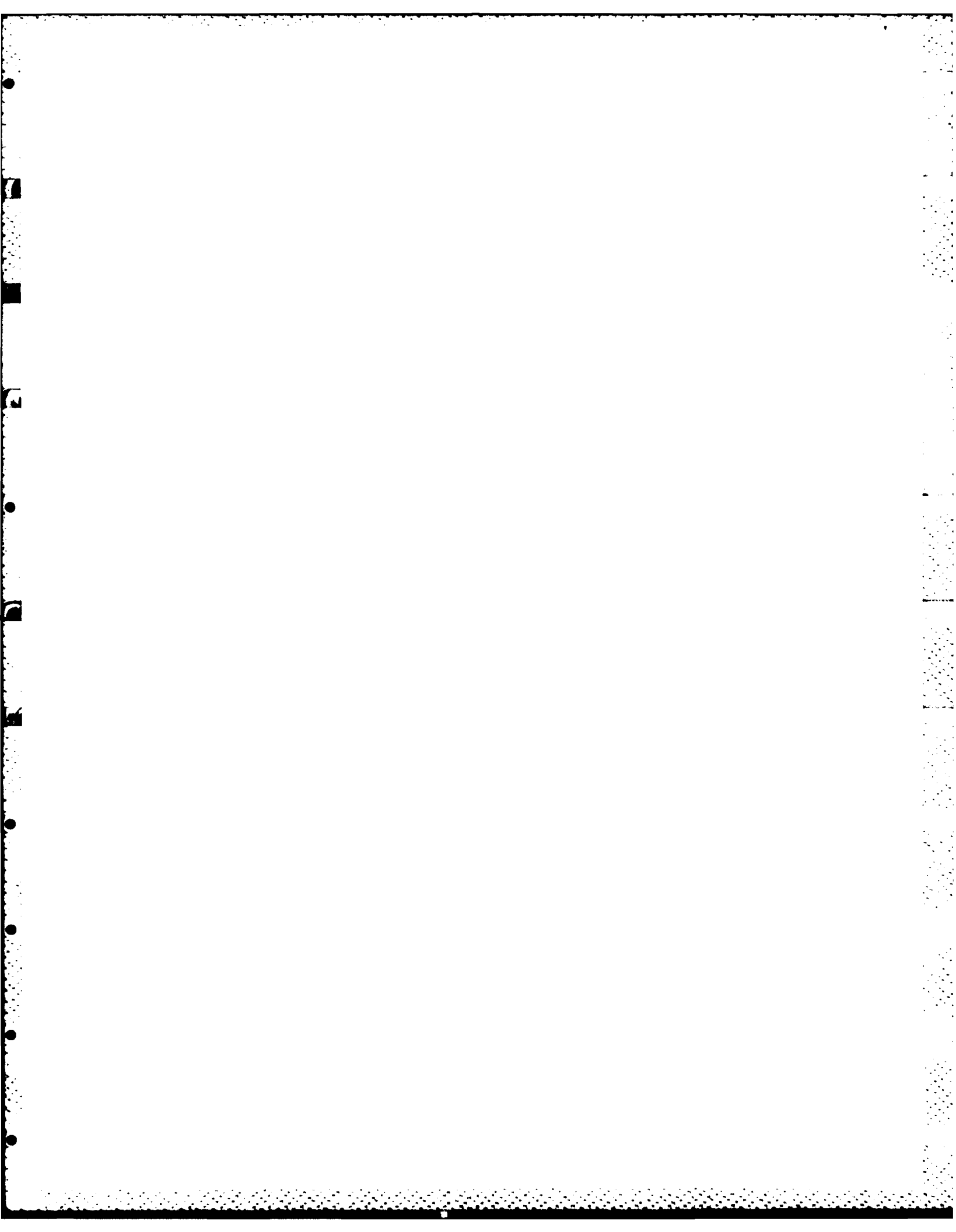
**Center for Research
in Computing Technology**

DTIC FILE COPY

This document has been approved
for release and distribution
in accordance with national
policy.



**Aiken Computation Laboratory
33 Oxford Street
Cambridge, Massachusetts 02138**



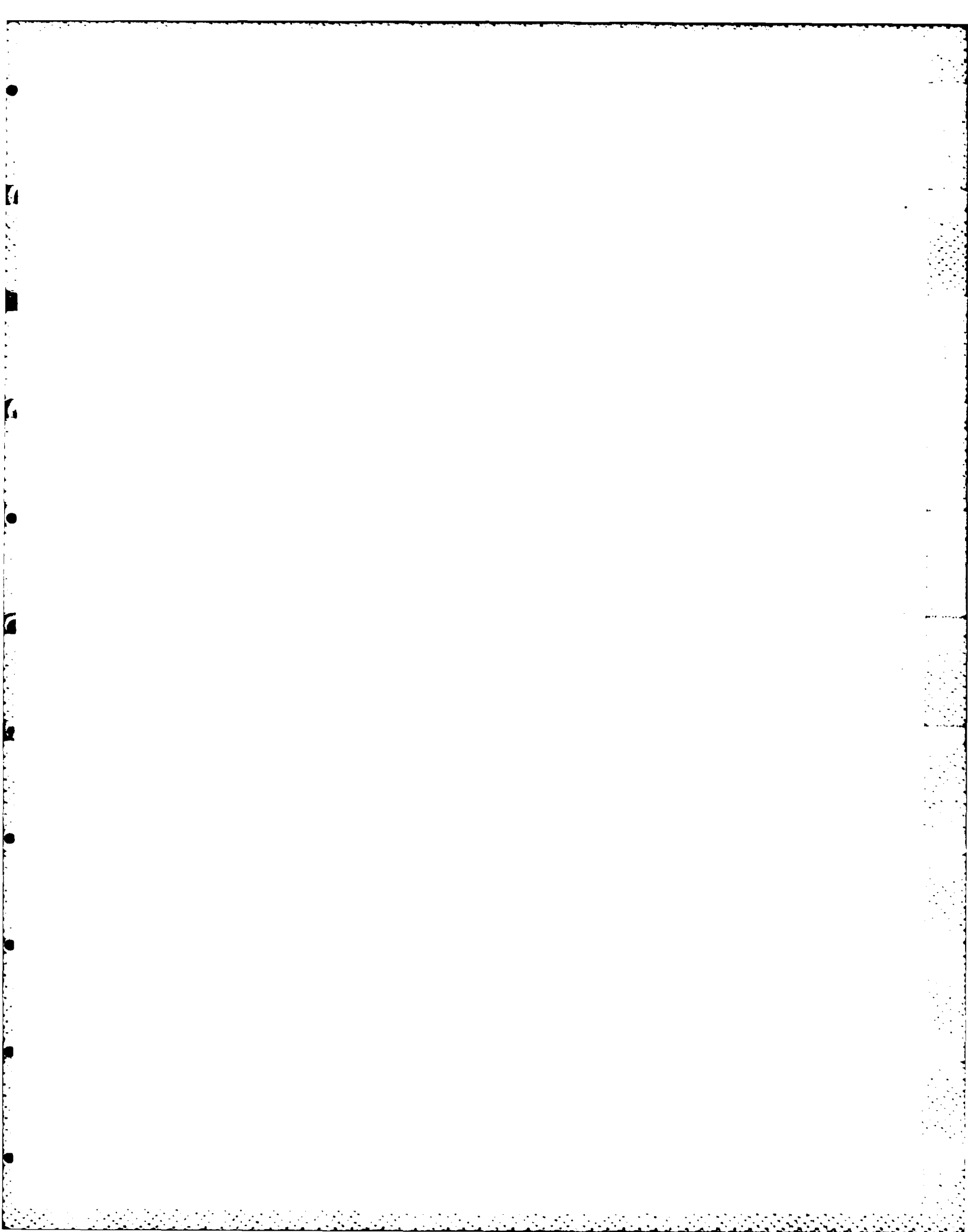
PARALLEL TIME $O(\log N)$ ACCEPTANCE
OF DETERMINISTIC CFLs

Philip N. Klein
John Reif

TR-05-84

March, 1984

DTIC
JUL 1984



unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTRIC CFLs		5. TYPE OF REPORT & PERIOD COVERED Technical Report
7. AUTHOR(s) John H. Reif Philip N. Klein		6. PERFORMING ORG. REPORT NUMBER TR-05-84
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Cambridge, MA 02138		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0647
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, VA 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same as above		12. REPORT DATE March 1984
		13. NUMBER OF PAGES 33
		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) unlimited		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) parallel algorithms, deterministic CFLs, language recognition		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) see reverse side.		

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

We give a parallel RAM algorithm for simulating a deterministic pushdown automaton. On an input of length n , a DPDA will use time $O(n)$; our simulation requires time $O(\log n)$ and only polynomially many processors. The algorithm can be easily adapted to accept the $LR(k)$ languages as well. An easy generalization of the algorithm will simulate a deterministic auxiliary pushdown automaton that uses space $s(n) \geq \log n$ and time $2^{O(s(n))}$; the simulation then requires time $O(s(n))$ and $2^{O(s(n))}$ processors. This simulation is nearly optimal for parallel RAMs, since we show that the language accepted in time $T(n)$ by a parallel RAM is accepted by a deterministic auxiliary pushdown automaton with space $T(n)$ and time $2^{O(T(n)^2)}$.

PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLs*

Philip N. Klein
John H. Reif

Aiken Computation Laboratory
Division of Applied Sciences
Harvard University
Cambridge, Massachusetts 02138

Accession For	
NTIS	✓
ERIC	
Biography	
Justification	
By	
Distribution	
Availability	
Dist	
A-1	

ERIC
COPY
1982/08

* This work was supported in part by National Science Foundation grant MCS-82-00269 and the Office of Naval Research contract N00014-80-C-0647.

A Preliminary version of this paper appeared in the IEEE Symposium on the Foundations of Computer Science, Chicago, IL, October 1982.

Abstract

We give a parallel RAM algorithm for simulating a deterministic pushdown automaton. On an input of length n , a DPDA will use time $O(n)$; our simulation requires time $O(\log n)$ and only polynomially many processors. The algorithm can be easily adapted to accept the $LR(k)$ languages as well. An easy generalization of the algorithm will simulate a deterministic auxiliary pushdown automaton that uses space $s(n) \geq \log n$ and time $2^{O(s(n))}$; the simulation then requires time $O(s(n))$ and $2^{O(s(n))}$ processors. This simulation is nearly optimal for parallel RAMs, since we show that the language accepted in time $T(n)$ by a parallel RAM is accepted by a deterministic auxiliary pushdown automaton with space $T(n)$ and time $2^{O(T(n)^2)}$.

Introduction

This paper assumes the parallel random access machine model *P-RAM* as defined in [Fortune and Wyllie, 78], which consists of a collection of synchronous deterministic unit-cost RAMs with shared memory locations indexed by the natural numbers. Simultaneous reads are allowed: on each step, any given memory location may be read simultaneously by any number of processors. However, no two distinct processors can attempt to simultaneously write into the same memory location on the same step.

A fundamental question, which we address in this paper, is the time complexity of the P-RAM: that is, what class of languages can be accepted by the P-RAM within a given time bound?

Previously [Fortune and Wyllie, 78] showed that any language accepted by a deterministic Turing machine with space bound $s(n) \geq \log n$ is accepted in time $O(s(n))$ by a P-RAM. Also [Ruzzo, 80] showed that any language accepted by an auxiliary pushdown machine with space $s(n) \geq \log n$ and time $2^{O(s(n))}$ is accepted in time $O(s(n))^2$ by various parallel machine models including the P-RAM.

The form of the paper is as follows: in Section 1, we will describe some assumptions we make of the simulated deterministic PDA in order to simplify the presentation of the algorithm. In Section 2, we introduce the notion of a surface configuration, and rephrase some of the assumptions of the first section as propositions about surface configurations. In Section 3, we introduce some notation useful in

the proof of the algorithm, and state some lemmas concerning this notation. In Section 4, we describe the algorithm, and, in Section 5, give an inductive proof of its correctness. In Section 6, we point out that the algorithm may be used to simulate a space-bounded auxiliary pushdown automaton. In Section 7, we give a complementary result concerning simulation of P-RAMs by deterministic auxiliary PDAs. In Section 7, we mention some related work, and in Section 8, we mention alternative models of parallel computation.

Section 1: Basic Assumptions

We denote the cardinality of a set A by $|A|$, and the length of a string s by $|s|$. We denote the empty string by ϵ . We denote the concatenation of strings s_1 and s_2 by $s_1 \circ s_2$.

We will simulate a deterministic PDA M with input alphabet Σ , state set Q , and stack alphabet S . We will for technical reasons assume that M 's transition function is a mapping from $(q, \omega, \sigma, \gamma)$ to (q', s) , where q is the current state and q' is the next state, ω is the input symbol currently scanned, σ is the top stack symbol (if it exists), s is a string of stack symbols of length at most 2 that will replace σ in the stack, and γ is *true* iff the stack consists only of σ . Thus M 's next move may depend on whether the stack has only one symbol. Note that standard techniques allow this assumption and those described below, and furthermore that any DPDA in the form defined in [Hopcroft and Ullman, pp. 253-255] may be transformed into one in the form used here without changing the language accepted.

For a fixed input $\omega_1 \dots \omega_n \in \Sigma^n$, we can write each configuration of M as $(q, i, s) \in Q \times \{0, \dots, n\} \times S^*$, where q is the state of the finite control, i is the position of the input head (i.e. the number of symbols that have been read), and s is the stack (with the rightmost symbol being the top of the stack). Let \vdash be the next move relation, and \vdash^+ and \vdash^* be its transitive and reflexive-transitive closures, respectively. For any non-negative integer a , let \vdash^a be the a -fold composition of \vdash ; i.e. reachability in exactly a steps.

We assume of the PDA that each push and pop changes the stack height by only one, that each push is accompanied by an advance of the input head, and that each advance of the input head is accompanied by a push. Any DCFI has such a PDA, which is essentially a shift-reduce parser for the DCFI. (see Hopcroft and Ullman, pp. 253-255). It follows that the height of the stack never grows by more than n , the length of the input.

We also assume that M never changes more than the top symbol of the stack: i.e., for $s, s' \in S^*$, $\sigma \in S$,

if $(q, i, s \circ \sigma) \vdash (q', i', s')$, then $s' = s \circ s''$ for some s'' .

We assume that there are no cycles $(q, i, s) \vdash^+ (q, i, s)$ of length more than one. Every such cycle can be replaced by a cycle of length one: $(q, i, s) \vdash (q, i, s)$. We let $\text{LOOP}(q, i, s)$ be the predicate that is *true* just in case $(q, i, s) \vdash (q, i, s)$.

We assume that M 's stack initially contains a special symbol, and this symbol is never popped (M accepts by final state rather than by empty stack). We may therefore require of M that when started in a configuration with a stack of size one, M will never pop that one symbol, but will loop instead. We have therefore

if $(q, i, s \circ \sigma) \vdash (q', i', s')$ then $|s'| \geq 1$.

Moreover, we assume that M never loops unless the stack has at most one symbol (i.e. M pops all but one of the stack symbols before looping):

if $\text{LOOP}(q, i, s)$ then $|s| = 1$.

We assume that the accepting configuration $(x_{\text{acc}}, \epsilon)$ is a looping configuration.

Finally, we assume that if not $\text{LOOP}(q, i, s)$ then the next move from (q, i, s) depends only on the top symbol of s :

for any $s_1, s_2 \in S^*$, $\sigma \in S$, if not $\text{LOOP}(q, i, s_1 \circ \sigma)$ and $(q, i, s_1 \circ \sigma) \vdash (q', i', s_1 \circ s')$,
then $(q, i, s_2 \circ \sigma) \vdash (q', i', s_2 \circ s')$.

Section 2: Surface Configurations

Instead of manipulating complete configurations of M , the algorithm manipulates surface configurations. A surface configuration does not specify the entire stack, only the top symbol. We also include in our surface configurations a stack height parameter, giving relative stack height information. In particular, for a fixed input $\omega_1 \dots \omega_n \in \Sigma^n$, we write each surface configuration as $(q, i, \sigma, h) \in Q \times \{0, \dots, n\} \times S \times \{0, \dots, n\}$, where q is the state of the finite control, i is the position of the input head, σ is the top symbol of the stack, and h is a stack height parameter. Let X be the set of all such surface configurations for input $\omega_1 \dots \omega_n$, and note that $|X| = O(n^2)$. We extend the definition of \vdash to apply to surface configurations as follows: if $(q, i, s, \sigma) \vdash (q', i', s', \sigma')$, then for any h we may write $((q, i, \sigma, h), s) \vdash ((q', i', \sigma', h'), s')$, where $h' = h + |s'| - |s|$. Because this is true for any h , we have the following proposition:

h-Independence Proposition: if $((q, i, \sigma, h), s) \vdash ((q', i', \sigma', h'), s')$, then for any δ , $((q, i, \sigma, h + \delta), s) \vdash ((q', i', \sigma', h' + \delta), s')$.

We define $h: X \rightarrow \{0, \dots, n\}$ by $h(q, i, \sigma, h) = h$. By a simple induction, we obtain:

Stack Height Proposition: For any $x, x' \in X$ and $s, s' \in S^*$, if $(x, s) \vdash^* (x', s')$ then $|s'| - |s| = h(x') - h(x)$.

Define $p: X \rightarrow \{0, \dots, n\}$ by $p(q, i, \sigma, h) = i$. That is, $p(x)$ gives the position of the input head in surface configuration x . Because the input head is one-way, we have:

Head Position Proposition I: If $(x, s) \vdash^* (x', s')$ then $p(x') \geq p(x)$.

Because we assumed each push is accompanied by an advance of the input head, we have:

Head Position Proposition II: if $(x, s) \vdash^* (x', s')$ and $|s'| > |s|$, then $p(x') > p(x)$.

We extend the definition of the LOOP predicate so that for any h , $\text{LOOP}((q, i, \sigma, h), s)$ iff $\text{LOOP}(q, i, s, \sigma)$.

Suppose M is in the configuration (x, s) , where x is a surface configuration. Because M never changes more than the top symbol of the stack, if M does not pop, then s remains intact. Because M 's next move depends only on the top symbol of the stack, x determines the next state. If M does not pop, then x also determines the new top stack symbol, and hence the next surface configuration. We have therefore:

Stack Proposition: if $(x, s) \vdash (x', s')$ and $|s'| \geq |s|$ then

- (1) $s' = s \circ s''$ for some $s'' \in S^*$, and
- (2) if not $\text{LOOP}(x, s)$ then for any $s_1 \in S^*$, $(x, s_1) \vdash (x', s_1 \circ s'')$.

Recalling that M only loops if its stack contains exactly one symbol:

LOOP Proposition: if $\text{LOOP}(x, s)$ then $s = \epsilon$.

By induction, we obtain:

Preservation of Time Proposition: if $(x, \epsilon) \vdash^a (x', s')$, but there is no $b < a$ such that $(x, \epsilon) \vdash^b (x', s')$, then for any s , $(x, s) \vdash^a (x', s \circ s')$.

Combining Computations Propositions:

- (1) if $(x, \epsilon) \vdash^* (x', s')$ and $(x', \epsilon) \vdash^* (x'', s'')$, then $(x, \epsilon) \vdash^* (x'', s' \circ s'')$.
- (2) if $(x, \epsilon) \vdash^* (x', \epsilon)$ and $(x', s) \vdash^* (x'', s'')$ then $(x, s) \vdash^* (x'', s'')$.

Section 3: Notation

We now introduce some notation that will be useful in proving the algorithm. We first need to prove a claim that will ensure the well-definedness of the notation:

claim if $(u, \epsilon) \vdash^* (x, s)$ and $(u, \epsilon) \vdash^* (x, s')$ then $s = s'$ (i.e. s is uniquely determined by u and x).

proof Either $(x, s) \vdash^* (x, s')$ or $(x, s') \vdash^* (x, s)$. Suppose without loss of generality that $(x, s) \vdash^*$

$$h(w) > h(u) = h(v).$$

lemma 17 Suppose $\{x, y, z\}_u$ and $\{z, y', z'\}_u$.

$$\text{Let } y'' = \begin{cases} y & \text{if } y \neq \perp \\ y' & \text{else} \end{cases}$$

Then $\{x, y'', z'\}_u$.

proof Since $\{x, z\}_u$ and $\{z, z'\}_u$ exist, so does $\{x, z'\}_u$.

case 1) $y \neq \perp$. Then for every $w \in \{x, y\}_u$ such that $w \neq u$ and $w \neq y$, $h(w) > h(u)$. Since

$y \in \{x, z'\}_u$, clearly $\{y, z'\}_u$ exists. We conclude $\{x, y, z'\}_u$.

case 2) $y = \perp$. Then for every $w \in \{x, z\}_u$ such that $w \neq u$ and $w \neq y$, $h(w) > h(u)$. Suppose

$y' \neq \perp$ (the case in which $y' = \perp$ is analogous). Then for every $w \in \{z, y'\}_u$ such that $w \neq u$ and $w \neq y'$, $h(w) > h(u)$. In summary, for every $w \in \{x, y'\}_u$ such that $w \neq u$ and $w \neq y'$, $h(w) > h(u)$. We also have that $\{y', z'\}_u$ exists, so $\{x, y', z'\}_u$.

lemma 18 If $[u]_v$, $h(u) > h(v)$, and $\{x, y\}_u$ exists, then $\{x, \perp, y\}_v$.

proof Suppose $w \in \{x, y\}_u$. Then $[w]_u$, so by the stack height proposition, $h(w) \geq h(u) > h(v)$. By

lemma 1, $\{x, y\}_v$ exists, so we may conclude that $\{x, \perp, y\}_v$.

case 2) $h(y') \leq h(y)$. In this case, $y'' = y'$. It follows from the hypothesis that if $(y', z)_{\mathcal{U}}$ exists and $w \in [y', z]_{\mathcal{U}}$, then $h(w) > h(y')$. It also follows from the hypothesis that $[x, z]_{\mathcal{U}}$ and $[z, y']_{\mathcal{U}}$ exist, so $[x, y']_{\mathcal{U}}$ exists. Suppose $w \in [x, y']_{\mathcal{U}}$; then either $w \in [x, z]_{\mathcal{U}}$ or $w \in [z, y']_{\mathcal{U}}$. If $w \in [x, z]_{\mathcal{U}}$ then $h(w) \geq h(y) \geq h(y')$. If $w \in [z, y']_{\mathcal{U}}$, then again $h(w) \geq h(y')$. We may therefore conclude $\langle x, y', z' \rangle_{\mathcal{U}}$.

lemma 7 If $[u]_{\mathcal{V}}$, $\langle x, y, a[z]_{\mathcal{U}} \rangle_{\mathcal{U}}$, and $h(y) > h(u)$, then $\langle x, y, a[z]_{\mathcal{V}} \rangle_{\mathcal{V}}$.

proof By the definition of $\langle x, y, a[z]_{\mathcal{U}} \rangle_{\mathcal{U}}$, $h(a[z]_{\mathcal{U}}) \geq h(y)$, so $h(a[z]_{\mathcal{U}}) > h(u)$, so $a[z]_{\mathcal{U}} = a[z]_{\mathcal{V}}$ by lemma

3. We therefore have $\langle x, y, a[z]_{\mathcal{V}} \rangle_{\mathcal{U}}$. But then by lemma 4, $\langle x, y, a[z]_{\mathcal{V}} \rangle_{\mathcal{V}}$.

lemma 15 If $h(w) > h(x)$ for every $w \in [x, y]_{\mathcal{U}}$ then $[y]_{\mathcal{X}}$.

proof Since $h(y) > h(x)$, clearly $y \neq x$. Since $[x, y]_{\mathcal{U}}$ exists, $(x, \epsilon) \vdash^* (x, s) \vdash^a (y, s')$ for some $s, s' \in S^*$,

and some minimal integer $a \geq 1$. We prove the lemma by induction on a . Consider the case

$a = 1$, so $(x, s) \vdash (y, s')$. Suppose $(x, \epsilon) \vdash (y', s'')$. Since $[x, y']_{\mathcal{X}}$ exists, by lemma 1

$[x, y']_{\mathcal{X}} = [x, y']_{\mathcal{U}} \subseteq [x, y]_{\mathcal{U}}$, so $h(y') > h(x)$, so $y' \neq x$. Then there is no $b < a = 1$ such that $(x, \epsilon) \vdash^b$

(y', s'') , so by the preservation of time proposition, $(x, s) \vdash (y', s \circ s'')$, so in fact $(y, s') = (y', s \circ s'')$.

We conclude that $[y]_{\mathcal{X}}$.

Now assume the lemma holds for $a \geq 1$, and suppose $(x, s) \vdash^a (x', s_1) \vdash (y, s')$. By the inductive hypothesis, $[x']_{\mathcal{X}}$, so $[x, x']_{\mathcal{X}}$ exists, so $h(x') > h(x)$ as above. Since $[x']_{\mathcal{X}}$, we have $(x, \epsilon) \vdash^a (x', s_2)$ for some $s_2 \in S^*$. Since $h(x') > h(x)$, by the stack height proposition $s_2 \neq \epsilon$, so by the LOOP proposition, not LOOP(x', s_2). Suppose $(x', s_2) \vdash (y', s'')$; we have that $(x, \epsilon) \vdash^{a+1} (y', s'')$ but there is no $b < a + 1$ such that $(x, \epsilon) \vdash^b (y', s'')$ (else $(x', s_2) = (y', s'')$, so LOOP(x', s_2)). Then by the preservation of time proposition, $(x, s) \vdash^{a+1} (y', s \circ s'')$, so $y = y'$, and we conclude that $[y]_{\mathcal{X}}$.

lemma 16 If $[u]_{\mathcal{V}}$, $h(u) = h(v)$, and $\{x, y, z\}_{\mathcal{U}}$, then $\{x, y, z\}_{\mathcal{V}}$.

proof Suppose $y = \perp$ (the case in which $y \neq \perp$ is analogous). Then $[x, z]_{\mathcal{U}}$ exists, and if $w \in [x, z]_{\mathcal{U}}$

then $h(w) > h(u)$. But by lemma 1, $[x, z]_{\mathcal{V}}$ exists and $[x, z]_{\mathcal{V}} = [x, z]_{\mathcal{U}}$, so if $w \in [x, z]_{\mathcal{V}}$ then

lemma 4 If $[u]_K$ and $\langle x, y, z \rangle_U$ then $\langle x, y, z \rangle_K$.

proof By lemma 1, $[x, y]_K$ exists and $[x, y]_K = [x, y]_U$. Then since $h(w) \geq h(y)$ for each $w \in [x, y]_U$, it follows that $h(w) \geq h(y)$ for each $w \in [x, y]_K$. Similarly, by lemma 2, if $(y, z)_U$ exists, then $(y, z)_K$ exists, and $(y, z)_K = (y, z)_U$. Hence if $w \in (y, z)_K$, then $w \in (y, z)_U$, so $h(w) > h(y)$. We conclude that $\langle x, y, z \rangle_K$.

lemma 5 If $(u, \epsilon) \vdash^* (x, s_1) \vdash^* (y, s_2)$ and for each $w \in [x, y]_U$, $h(w) \geq h(y)$, then $s_1 = s_2 \circ s'$ for some $s' \in S^*$.

proof The lemma follows from the following claim:

claim if $(u, \epsilon) \vdash^* (x, s_1) \vdash^a (y, s_2)$ for some a , then for any h such that $h(w) \geq h \geq h(u)$ for every $w \in [x, y]_U$, there is a string of length $h - h(u)$ that is an initial substring of s_1 and s_2 .

proof of claim by induction on a . The case $a = 0$ is trivial, so assume the claim is true for a , and suppose $(u, \epsilon) \vdash^* (x, s_1) \vdash (x', s_1') \vdash^a (y, s_2)$. By the inductive hypothesis, there is a string s of length $h - h(u)$ that is an initial substring of s_1' and s_2 . Consider the move $(x, s_1) \vdash (x', s_1')$.

If the move is a pop, then s_1' is an initial substring of s_1 , hence so is s . Therefore, assume the move is not a pop, so by the stack proposition, $s_1' = s_1 \circ s''$ for some $s'' \in S^*$. Now by the stack height proposition, $|s_1| = h(x) - h(u)$. Since clearly $x \in [x, y]_U$, we have $h(x) \geq h$. Therefore, $h - h(u) \leq h(x) - h(u)$, so s is no longer than s_1 . Since s is an initial substring of $s_1' = s_1 \circ s''$, s is an initial substring of s_1 .

lemma 6 Suppose $\langle x, y, z \rangle_U$ and $\langle z, y', z' \rangle_U$.

Let $y'' = \begin{cases} y' & \text{if } h(y') \leq h(y) \\ y & \text{else} \end{cases}$

Then $\langle x, y'', z' \rangle_U$.

proof

case 1) $h(y') > h(y)$. In this case, $y'' = y$. It follows from the hypothesis that $[x, y]_U$ exists and $h(w) \geq h(y)$ for every $w \in [x, y]_U$. Suppose $(y, z')_U$ exists, and $w \in (y, z')_U$. Either $w \in (y, z)_U$ or $w \in (z, z')_U$. If $w \in (y, z)_U$ then $h(w) > h(y)$ follows from the hypothesis. If $w \in (z, z')_U$, then $h(w) \geq h(y') > h(y)$. Since in either case $h(w) > h(y)$, we may conclude $\langle x, y, z' \rangle_U$.

Appendix

lemma 0 Suppose $(x, \epsilon) \vdash^a (x', s')$, but there is no $b < a$ such that $(x, \epsilon) \vdash^b (x', s')$. Then for any $s \in S^*$

and each $0 \leq b < a$,

(i) if $(x, \epsilon) \vdash^b (x'', s'')$ then $(x, s) \vdash^b (x'', s \circ s'')$, and

(ii) if $(x, s) \vdash^b (x'', s_2)$ then $(x, \epsilon) \vdash^b (x'', s'')$ with $s_2 = s \circ s''$.

proof By the hypothesis, if $(x, \epsilon) \vdash^b (x'', s'')$ then not $\text{LOOP}(x'', s'')$. Then (i) follows by

Preservation of Time Proposition. Now suppose $(x, s) \vdash^b (x'', s_2)$. Choose (x_2, s'') so that (x, ϵ)

$\vdash^b (x_2, s'')$. Again not $\text{LOOP}(x_2, s'')$, so $(x, s) \vdash^b (x_2, s \circ s'')$, so $(x_2, s \circ s'') = (x'', s_2)$.

lemma 1 If $[u]_V$ and $[x, y]_U$ exists, then $[x, y]_V$ exists, and $[x, y]_V = [x, y]_U$.

proof $(v, \epsilon) \vdash^* (u, s)$ and $(u, \epsilon) \vdash^* (x, s_1) \vdash^* (y, s_2)$ for some $s, s_1, s_2 \in S^*$. Hence $(v, \epsilon) \vdash^* (u, s) \vdash^*$

$(x, s \circ s_1) \vdash^* (y, s \circ s_2)$, so $[x, y]_V$ exists. Moreover, if $w \in [x, y]_U$ then $(x, s_1) \vdash^* (w, s_3) \vdash^* (y, s_2)$ for

some $s_3 \in S^*$, so $(x, s \circ s_1) \vdash^* (w, s \circ s_3) \vdash^* (y, s \circ s_2)$, so $w \in [x, y]_V$. Now, let a be minimal such that

$(u, \epsilon) \vdash^a (y, s_2)$, and suppose $w \in [x, y]_V$. Then $(u, s) \vdash^b (w, s_3) \vdash^c (y, s \circ s_2)$ for some s_3 and

some b, c such that $b + c \leq a$. If $(w, s_3) = (y, s \circ s_2)$, then $w = y \in [x, y]_U$. Otherwise, by lemma 0,

$(u, \epsilon) \vdash^b (w, s')$ with $s_3 = s \circ s'$. Because $(u, \epsilon) \vdash^* (x, s_1)$, either $(x, s_1) \vdash^* (w, s')$ or $(w, s') \vdash^+$

(x, s_1) . In the former case, $w \in [x, y]_U$. In the latter case, $(w, s_3) \vdash^+ (x, s \circ s_1)$, contradicting

choice of w .

lemma 2 If $[u]_V$ and $(x, y)_U$ exists, then $(x, y)_V$ exists, and $(x, y)_V = (x, y)_U$.

proof immediate from lemma 1.

lemma 3 If $[u]_V$ and $h(a[x]_U) > h(u)$, then $a[x]_V = a[x]_U$.

proof By definition of $a[x]_U$, $(u, \epsilon) \vdash^* (x, s_1) \vdash^a (a[x]_U, s_2)$ for some $s_1, s_2 \in S^*$. Because $h(a[x]_U) >$

$h(u) > 0$, $|s_2| - |\epsilon| = |s_2| > 0$ by Stack Height Proposition, so $s_2 \neq \epsilon$, so not $\text{LOOP}(a[x]_U, s_2)$ by the

LOOP Proposition. Then by Preservation of Time Proposition, $(u, s) \vdash^* (x, s \circ s_1) \vdash^a$

$(a[x]_U, s \circ s_2)$ for any $s \in S^*$. In particular, if s satisfies $(v, \epsilon) \vdash^* (u, s)$, then $(v, \epsilon) \vdash^* (x, s \circ s_1) \vdash^a$

$(a[x]_U, s \circ s_2)$, so by definition, $a[x]_V = a[x]_U$.

$O(1(n))$. Thus only $1(n)$ space is required by our simulating deterministic APDA.

Note: [Ruzzo, 82] has also proved this result independently by combining some known complexity bounds for various parallel machine models.

Section 8: Further Work

The LR(k) grammars considered in [Knuth, 65] are frequently used in practice for programming languages. The k -symbol lookahead required for recognition of an LR(k) language by a DPDA may be incorporated into stage 0 of our algorithm, using only $O(k)$ extra time.

As mentioned at the end of Section 4, our algorithm for DCFLL recognition requires time $O(\log n)$ and $O(n^4)$ processors. It is easy to show that in fact only $O(n^3)$ processors are necessary, after a minor modification of the algorithm.

Section 9: Alternative Parallel Machine Models

[Ruzzo, 80] gave an alternating machine algorithm for recognition of context-free languages in time $O(\log^2 n)$ and simultaneously polynomial tree-size. As he points out, this algorithm can easily be simulated in time $O(\log n)$ and a polynomial number of processors by parallel machine models which allow resolution of both read and write conflicts. It is also easy to show that Ruzzo's algorithm can be simulated by a depth $O(\log n)$ circuit with a polynomial number of logical elements but unbounded degree. (See [Stockmeyer and Vishkin, 81].)

However, Ruzzo's algorithm requires $\Omega(\log^2 n)$ time on the usual models of parallel computation that disallow write conflicts. It is an open question whether general context-free recognition can be done in time $o(\log^2 n)$ on a P-RAM. Also, it is open whether circuits of constant degree and depth $o(\log^2 n)$ can recognize the class of languages accepted by deterministic TMs with space $O(\log n)$.

Theorem 2 Let L be accepted by a DPDA. Then L is accepted by a P-RAM with time $O(\log n)$ and $O(n^4)$ processors.

proof The theorem follows from Theorem 1 and comments at the end of section 4.

Section 6: Simulation of a Deterministic Auxiliary Pushdown Automaton

Consider now the simulation of an $s(n)$ space-bounded, $t(n)$ time-bounded deterministic auxiliary pushdown automaton M with a stack discipline satisfying the assumptions of Section 1. Each surface configuration for such a machine will contain:

- (i) the current state of M (in the finite control),
- (ii) the position of the input head,
- (iii) the contents of the work tapes and positions of the work tape heads,
- (iv) the relative height parameter, which may be bounded by the time bound $t(n)$.

For a fixed machine M , the number of such surface configurations is bounded by $2^{O(s(n))} \cdot t(n)^{O(1)}$.

The simulation algorithm is exactly the one given in section 4, except that the number of stages is now $\lceil \log t(n) \rceil + 1$. Thus, if $t(n) = 2^{O(s(n))}$, the algorithm requires time $O(s(n))$ and $2^{O(s(n))}$ processors. Thus we have

Theorem 3 Let L be accepted by a deterministic APDA with space $s(n)$ and time $2^{O(s(n))}$. Then L is accepted by a P-RAM with time $O(s(n))$ and $2^{O(s(n))}$ processors.

Section 7: Simulation of P-RAMs by Deterministic APDAs

We show here that our P-RAM algorithm of Section 4 for simulating deterministic APDAs is nearly optimal, since there is a complementing simulation of P-RAMs by deterministic APDAs.

Theorem 4 Let L be accepted by P-RAM with time $T(n)$. Then L is accepted by a deterministic APDA with space $T(n)$ and time $2^{O(T(n)^2)}$.

proof [Fortune and Wyllie, 78] prove in their Lemma 1b that L is accepted by a deterministic TM with $T(n)^2$ space, and time $2^{O(T(n)^2)}$. We use exactly their algorithm, but implement it on a deterministic APDA. Their algorithm is recursive and requires a pushdown stack of size at most $T(n)$, where each element on the stack can be represented by a bit sequence of length

conclude $\langle y, z, (2^{k+1})y \rangle_x$.

lemma 13 If $\langle P_{k+1}[x], y, y \rangle_x$ then $\langle x, \text{PREDICT}_{k+1}[x, y], (2^{k+1})y \rangle_x$. (Correctness of PREDICT_{k+1}).

proof Let $x^* = R_k[x]$.

case 1) $h(y) \leq h(x^*)$. In this case, $\text{PREDICT}_{k+1}[x, y] = \text{HOP}_{k+1}[x, y]$. [see figure 12a] By

corollary to lemma 10, $\langle P_k[x], I_k[x^*], P_{k+1}[x] \rangle_x$, and we assume $\langle P_{k+1}[x], y, y \rangle_x$.

Since $h(y) \leq h(x^*) = h(I_k[x^*])$, by lemma 6, $\langle P_k[x], y, y \rangle_x$. Then by lemma 12,

$\langle y, \text{HOP}_{k+1}[x, y], (2^{k+1})y \rangle_x$.

case 2) $h(y) > h(x^*)$. We are given $\langle P_{k+1}[x], y, y \rangle_x$. Choose b so that $y = b[P_{k+1}[x]]_x$.

Recall that $P_{k+1}[x] = P_k[x^*]$, so $[P_{k+1}[x]]_x = x^*$, and let $y' = b[P_{k+1}[x]]_x = x^*$. We have

$h(P_{k+1}[x]) \geq h(y) > h(x^*)$, so by lemma 3, $y' = b[P_{k+1}[x]]_x = b[P_{k+1}[x]]_x = y$. Then

$[P_{k+1}[x], y]_{x^*}$ exists, so by lemma 1, $[P_{k+1}[x], y]_{x^*} = [P_{k+1}[x], y]_x$. Now suppose

$w \in [P_{k+1}[x], y]_{x^*}$: then $w \in [P_{k+1}[x], y]_x$, so $h(w) \geq h(y)$ (since we have

$\langle P_{k+1}[x], y, y \rangle_x$.) We may then conclude $\langle P_k[x^*], y, y \rangle_{x^*}$ (recalling that

$P_k[x^*] = P_{k+1}[x]$). Let $z = \text{HOP}_{k+1}[x^*, y]$. By lemma 12, $\langle y, z, (2^{k+1})y \rangle_{x^*}$.

case A) $h(z) > h(x^*)$. [see figure 12b] In this case, $\text{PREDICT}_{k+1}[x, y] = z$, and by

lemma 7, $\langle y, z, (2^{k+1})y \rangle_x$.

case B) $h(z) = h(x^*)$. In this case, $\text{PREDICT}_{k+1}[x, y] = \text{HOP}_{k+1}[x, z]$. [see figure

12c] Since $\langle P_{k+1}[x], y, y \rangle_x$ and $\langle y, z, z \rangle_x$, by lemma 6 we have

$\langle P_{k+1}[x], z, z \rangle_x$. Let $z' = \text{HOP}_{k+1}[x, z]$. By lemma 12, $\langle z, z', (2^{k+1})z \rangle_x$.

Then certainly $\langle z, z', (2^{k+1})z \rangle_x$ and so, by lemma 6, $\langle y, z', (2^{k+1})y \rangle_x$.

lemma 14 $\langle P_{k+1}[x], R_{k+1}[x], (2^{k+1})P_{k+1}[x] \rangle_x$. (Correctness of R_{k+1})

proof By definition, $R_{k+1}[x] = \text{PREDICT}_{k+1}[x, P_{k+1}[x]]$. Trivially,

$\langle P_{k+1}[x], P_{k+1}[x], P_{k+1}[x] \rangle_x$, so the lemma follows from lemma 13.

Theorem 1 follows by induction on k .

corollary $\langle P_k[x], l_k[R_k[x]], P_{k+1}[x] \rangle_x$.

proof Let $x^* = R_k[x]$. By correctness of R_k , $\langle P_k[x], x^*, x^* \rangle_x$. By lemma 10, $\langle P_k[x], l_k[x^*], P_k[x^*] \rangle_x$.

Recalling that $P_{k+1}[x] = P_k[x^*]$, we are done.

lemma 11 $\langle x, l_{k+1}[x], P_{k+1}[x] \rangle_x$ and $h(l_{k+1}[x]) = h(x)$. (Correctness of l_{k+1})

proof [see figure 10] Clearly $h(l_{k+1}[x]) = h(x)$ by the definition of $l_{k+1}[x]$. By correctness of

l_k , $\langle x, l_k[x], P_k[x] \rangle_x$. By corollary to lemma 10, $\langle P_k[x], l_k[R_k[x]], P_{k+1}[x] \rangle_x$. Then by lemma

6, $\langle x, l_{k+1}[x], P_{k+1}[x] \rangle_x$ (see definition of $l_{k+1}[x]$).

lemma 12 If $\langle P_k[x], y, y \rangle_x$ then $\langle y, HOP_{k+1}[x, y], (2^{k+1})y \rangle_x$.

proof Let $z = \text{PREDICT}_k[x, y]$. Since $\langle P_k[x], y, y \rangle_x$, by correctness of PREDICT_k , $\langle y, z, (2^k)y \rangle_x$.

It follows that $\langle y, z, z \rangle_x$, and that $[y, z]_x$ exists. Let $z^* = R_k[z]$. By correctness of R_k ,

$\langle P_k[z], z^*, (2^k)P_k[z] \rangle_z$.

case 1) $h(z^*) > h(z)$. In this case, $HOP_{k+1}[x, y] = L_k[z]$. [see figure 21a] By lemma 7,

$\langle P_k[z], z^*, (2^k)P_k[z] \rangle_x$. Since $\langle y, z, z \rangle_x$, we may apply lemma 10 to obtain

$\langle y, L_k[z], P_k[z] \rangle_x$. Hence, by lemma 6, since $h(L_k[z]) = h(z) < h(z^*)$, we obtain

$\langle y, L_k[z], (2^k)P_k[z] \rangle_x$. By correctness of P_k , $P_k[z] = a[z]_z$ for some $a \geq 2^k$. Since

$h(P_k[z]) \geq h(z^*) > h(z)$, by lemma 3 we obtain $a[z]_z = a[z]_x$, hence $P_k[z] = a[z]_x$. Since

$[y, z]_x$ exists, $z = b[y]_x$ for some $b \geq 0$, so $(2^k)P_k[z]_x = (2^k + a + b)y_x$. Since $a \geq 2^k$,

$(2^k)P_k[z]_x = (2^{k+1} + c)y_x$ for some $c \geq 0$, so we may conclude

$\langle x, L_k[z], (2^{k+1})y \rangle_x$.

case 2) $h(z^*) = h(z)$. In this case, $HOP_{k+1}[x, y] = \text{PREDICT}_k[x, z^*]$. [see figure 21b] We

assumed $\langle P_k[x], y, y \rangle_x$. By correctness of PREDICT_k , $\langle y, z, z \rangle_x$. By the stack height

proposition, $\langle z, z^*, z^* \rangle_z$ so by lemma 4, $\langle z, z^*, z^* \rangle_x$. Then by lemma 6, $\langle y, z^*, z^* \rangle_x$, and

by a second application of lemma 6, $\langle P_k[x], z^*, z^* \rangle_x$. Let $z' = \text{PREDICT}_k[x, z^*]$. By

correctness of PREDICT_k , $\langle z^*, z', (2^k)z^* \rangle_x$. Since $\langle y, z^*, z^* \rangle_x$, by lemma 6 we obtain

$\langle y, z', (2^k)z^* \rangle_x$.

If $z^* \in (z, (2^k)y]_x$, then z^* would contradict $\langle y, z, (2^k)y \rangle_x$ because $h(z^*) = h(z)$.

Thus $z^* = a[y]_x$ for some $a \geq 2^k$, so $(2^k)z^* = b[y]_x$ for some $b \geq 2^{k+1}$, so we may

hold for $k+1$. Then by induction, $P_{\lceil \log T \rceil}$ satisfies its correctness hypothesis. In particular, if x_{init} is the initial surface configuration for input $\omega_1 \dots \omega_n$, $P_{\lceil \log T \rceil}[x_{\text{init}}]$ will give the final surface configuration, and thus the final state of the PDA. The proof follows:

Theorem 1 $(x, \epsilon) \vdash^a (P_{\lceil \log T \rceil}[x], s_{\lceil \log T \rceil}[x])$ for $a \geq T$.

lemma 8 Let $x^* = R_k[x]$. Then $(x, \epsilon) \vdash^* (x^*, s_k[x, x^*])$.

proof [see figure 17] By correctness of P_k , $(x, \epsilon) \vdash^* (P_k[x], s_k[x])$. By correctness of R_k , $(P_k[x], s_k[x]) \vdash^* (x^*, s)$ for some s , and if $w \in [P_k[x], x^*]_x$ then $h(w) \geq h(x^*)$. Hence by lemma 5, s is an initial substring of $s_k[x]$. Since by stack height proposition, $|s| = h(x^*) - h(x)$, we conclude $s = s_k[x, x^*]$.

lemma 9 $(x, \epsilon) \vdash^a (P_{k+1}[x], s_{k+1}[x])$ for some $a \geq 2^{k+1}$. (Correctness of P_{k+1})

proof Let $x^* = R_k[x]$. By correctness of P_k , $(x, \epsilon) \vdash^a (P_k[x], s_k[x])$ and $(x^*, \epsilon) \vdash^b (P_k[x^*], s_k[x^*])$ for some $a, b \geq 2^k$.

By lemma 8, $(x, \epsilon) \vdash^* (x^*, s_k[x, x^*])$, so by combining computations, $(x, \epsilon) \vdash^*$

$(P_k[x^*], s_k[x, x^*] \circ s_k[x^*])$, and $(P_k[x^*], s_k[x, x^*] \circ s_k[x^*]) = (P_{k+1}[x], s_{k+1}[x])$ by definition of P_{k+1} and s_{k+1} .

case 1) $h(P_k[x^*]) = h(x^*)$. [see figure 18] By correctness of R_k , if $w \in (x^*, (2^k)[P_k[x]]_x)$ then $h(w) > h(x^*)$, so we must conclude that $P_k[x^*] \notin (x^*, (2^k)[P_k[x]]_x)$, so $(P_k[x], s_k[x]) \vdash^c (P_k[x^*], s_{k+1}[x])$ for some $c > 2^k$.

case 2) $h(P_k[x^*]) > h(x^*)$. [see figure 19] Then by lemma 3, $(x^*, s_k[x, x^*]) \vdash^b$

$(P_k[x^*], s_k[x, x^*] \circ s_k[x^*])$, so again $(P_k[x], s_k[x]) \vdash^c (P_k[x^*], s_{k+1}[x])$ for some $c \geq b \geq 2^k$.

We conclude in either case that $(x, \epsilon) \vdash^{a+c} (P_{k+1}[x], s_{k+1}[x])$ with $a+c \geq 2^k + 2^k = 2^{k+1}$.

lemma 10 If $\langle x, y, y \rangle_u$ then $\langle x, L_k[y], P_k[y] \rangle_u$.

proof Let $y' = L_k[y]$. [see figure 20] By correctness of L_k , $\langle y, y', P_k[y] \rangle_y$ and $h(y') = h(y)$. Since

$[y]_u, \langle y, y', P_k[y] \rangle_u$ by lemma 4. Then by lemma 6, we conclude $\langle x, y', P_k[y] \rangle_u$, since

$h(y') = h(y)$.

Section 5: Proof of Correctness

We introduce variables denoting the contents of the simulated stack. Their values are never computed; they exist only to make the proof easier. We define $s_k[x]$ and $s_k[x,y]$ inductively as follows:

if $(x, \epsilon) \vdash (y, s)$

then let $s_0[x] = s$

For $k = 0, \dots, \lceil \log T \rceil - 1$,

let $s_{k+1}[x] = s_k[x, x^*] \circ s_k[x^*]$

where $x^* = R_k[x]$

and for all $y \in X$ with $h(y) \geq h(x)$,

$s_k[x, y] =$ the leftmost $h(y) - h(x)$ symbols of $s_k[x]$.

The values of the arrays $P_k[x]$, $L_k[x]$, $HOP_k[x, y]$, $PREDICT_k[x, y]$, and $R_k[x]$ will be inductively shown to satisfy the following correctness hypotheses, for each k :

Correctness of P_k : $(x, \epsilon) \vdash^a (P_k[x], s_k[x])$ for some $a \geq 2^k$. [see figure 13]

Correctness of L_k : $\langle x, L_k[x], P_k[x] \rangle_x$ and $h(L_k[x]) = h(x)$.

[see figure 14]

Correctness of $PREDICT_k$: if $\langle P_k[x], y, y \rangle_x$ then $\langle y, PREDICT_k[x, y], (2^k)y \rangle_x$.

[see figure 15]

Correctness of R_k : $\langle P_k[x], R_k[x], (2^k)[P_k[x]]_x \rangle_x$.

[see figure 16]

It can easily be checked that the initial values satisfy the above hypotheses, for $k = 0$. Assuming the correctness hypotheses hold for P_k , L_k , $PREDICT_k$, and R_k , we show the corresponding conditions

For each $x, y \in X$ such that $h(x) \leq h(y) \leq h(P_k[x])$,

$$\text{let } \text{HOP}_{k+1}[x, y] := \begin{cases} \text{PREDICT}_k[x, z^*] & \text{if } h(z^*) = h(z) \\ P_k[z] & \text{else} \end{cases}$$
 where $z = \text{PREDICT}_k[x, y]$ and $z^* = R_k[z]$
 [see figure 11]

For each $x, y \in X$ such that $h(x) \leq h(y) \leq h(P_{k+1}[x])$,

$$\text{let } \text{PREDICT}_{k+1}[x, y] := \begin{cases} \text{HOP}_{k+1}[x, y] & \text{if } h(y) \leq h(x^*) \\ \text{HOP}_{k+1}[x, z] & \text{if } h(y) > h(x^*) \text{ and } h(z) = h(x^*) \\ z & \text{else} \end{cases}$$
 where $x^* = R_k[x]$ and $z = \text{HOP}_{k+1}[x^*, y]$
 [see figure 12]

For each $x \in X$,

$$\text{let } R_{k+1}[x] := \text{PREDICT}_{k+1}[x, P_{k+1}[x]]$$

We shall show in the next section that $P_k[x]$ gives a surface configuration reachable from (x, ϵ) in at least 2^k steps. In particular, if $(x_{\text{init}}, \epsilon)$ is the initial configuration and $(x_{\text{acc}}, \epsilon)$ the accepting configuration when the input is $\omega_1 \dots \omega_n \in \Sigma^n$, then $P_{\lceil \log T \rceil}[x_{\text{init}}] = x_{\text{acc}}$ iff M accepts the input (recalling that we assumed $(x_{\text{acc}}, \epsilon)$ to be a looping configuration).

Each of the $\lceil \log T \rceil + 1$ stages requires constant time on a P-RAM, if we assign a processor to each pair x, y of surface configurations. Recalling that $T = 2|Q||S|(n+1)$ and that the number of surface configurations is $|X| = |Q||S|(n+1)^2$, we see that, for a fixed DPDA, the algorithm requires parallel time $O(\log n)$ and $O(n^4)$ processors.

reader scan the inductive corrective hypotheses and the corresponding figures (found in the next section) in order to better understand the algorithm.

The initialization is as follows.

For each $x \in X$,

if $(x, \epsilon) \vdash (y, s)$,

let $P_0[x] := y$

let $L_0[x] := \begin{cases} P_0[x] & \text{if } h(P_0[x]) = 0 \\ x & \text{else} \end{cases}$

For each $x, y \in X$ such that $h(x) \leq h(y) \leq h(P_0[x])$,

if $(x, \epsilon) \vdash (x', s')$, s is the initial substring of s' of length $h(y) - h(x)$,

and $(y, s) \vdash (z, s'')$ for some s'' ,

let $\text{PREDICT}_0[x, y] := \begin{cases} z & \text{if } h(z) \leq h(y) \\ y & \text{else} \end{cases}$

[see figure 8]

For each $x \in X$,

let $R_0[x] := \text{PREDICT}_0[x, P_0[x]]$

The algorithm proceeds in stages $k = 0, \dots, \lceil \log T \rceil$. At stage $k + 1$, we assume that the values of P_k , L_k , PREDICT_k , and R_k have been stored, and compute P_{k+1} , L_{k+1} , PREDICT_{k+1} , and R_{k+1} as follows:

For each $x \in X$,

let $P_{k+1}[x] := P_k[R_k[x]]$

[see figure 9]
let $L_{k+1}[x] := \begin{cases} L_k[R_k[x]] & \text{if } h(R_k[x]) = h(x) \\ L_k[x] & \text{else} \end{cases}$

[see figure 10]

lemma 1 If $[u]_V$ and $[x,y]_U$ exists, then $[x,y]_V$ exists, and $[x,y]_V = [x,y]_U$.

[see figure 2]

lemma 2 If $[u]_V$ and $(x,y)_U$ exists, then $(x,y)_V$ exists, and $(x,y)_V = (x,y)_U$.

lemma 3 If $[u]_V$ and $h(a[x]_U) > h(u)$, then $a[x]_V = a[x]_U$.

[see figure 3]

lemma 4 If $[u]_V$ and $\langle x,y,z \rangle_U$, then $\langle x,y,z \rangle_V$.

[see figure 4]

lemma 5 If $(u,\epsilon) \vdash^* (x,s_1) \vdash^* (y,s_2)$ and for each $w \in [x,y]_U$, $h(w) \geq h(y)$, then $s_1 = s_2 \circ s'$ for some $s' \in S^*$.

[see figure 5]

lemma 6 Suppose $\langle x,y,z \rangle_U$ and $\langle z,y',z' \rangle_U$.

$$\text{Let } y'' = \begin{cases} y' & \text{if } h(y') \leq h(y) \\ y & \text{else} \end{cases}$$

Then $\langle x,y'',z' \rangle_U$.

[see figure 6]

lemma 7 If $[u]_V$, $\langle x,y,a[z]_U \rangle_U$, and $h(y) > h(u)$, then $\langle x,y,a[z]_V \rangle_V$.

[see figure 7]

Section 4: The Algorithm

We now describe the algorithm for simulating $T = 2|Q||S|(n+1)$ steps of M . (An argument like that found in [Aho and Ullman, p. 396] will establish that if M accepts an input of length n , it does so in at most T steps.) The data structures used are arrays indexed by surface configurations and having surface configurations as their elements: $P_k[x]$, $I_k[x]$, $HOP_k[x,y]$, $PREDICT_k[x,y]$, and $R_k[x]$, for $k = 0, \dots, \lceil \log T \rceil$. The algorithm is relatively short, but its proof contains a non-trivial induction. We suggest the

(x, s') . By the stack height proposition, $|s'| = |s|$. If any intermediate configuration (y, s'') had $|s''| > |s|$ then $p(x) \geq p(y) > p(x)$, a contradiction. Similarly, if $|s''| < |s| = |s'|$, then $p(x) > p(y) \geq p(x)$. Hence $|s''| = |s|$; i.e. every intermediate configuration has stack height equal to $|s|$. But since M never changes more than the top symbol of the stack, we may conclude $s' = s$.

Let $[x]_u$ denote the proposition $(u, \epsilon) \vdash^* (x, s)$ for some s .

If $(u, \epsilon) \vdash^* (x, s_1) \vdash^* (y, s_2)$ for some s_1, s_2 , then we shall say $[x, y]_u$ exists, and denotes the set $\{z \mid (x, s_1) \vdash^* (z, s_3) \vdash^* (y, s_2) \text{ for some } s_3\}$; otherwise $[x, y]_u$ does not exist.

If $[x, y]_u$ exists, we let $(x, y)_u = \{z \mid z \in [x, y]_u \text{ and } z \neq x\}$.

Note that $[x, y]_u$ and $(x, y)_u$ are intended to suggest closed and open intervals, respectively.

If $(u, \epsilon) \vdash^* (x, s_1) \vdash^a (y, s_2)$ then $a[x]_u$ denotes y .

Let $\langle x, y, z \rangle_u$ denote the proposition:

- (1) $[x, y]_u$ and $[x, z]_u$ exist;
- and (2) if $w \in [x, y]_u$ then $h(w) \geq h(y)$,
- and (3) if $(y, z)_u$ exists and $w \in (y, z)_u$ then $h(w) > h(y)$.

Informally, with respect to computations starting at u , either y occurs between x and z and has minimal stack height of all such intermediate configurations (and is otherwise latest), or y occurs after z and has minimal stack height of all configurations between x and z . [see figure 1]

Note that it follows from $\langle x, y, z \rangle_u$ that $\langle x, y, w \rangle_u$ for any $w \in [y, z]_u$ (and in particular that $\langle x, y, y \rangle_u$), and that $\langle w, y, z \rangle_u$ for any $w \in [x, y]_u$ (and in particular that $\langle y, y, z \rangle_u$).

We state eight lemmas whose proofs may be found in the appendix. We suggest the reader try proving one in order to become familiar with the notation.

REFERENCES

- Aho, A. and J. D. Ullman, *The theory of Parsing, Translation, and Compiling; volume 1: Parsing*, Prentice-Hall, (1972).
- Braunmuhl, B. von and R. Verbeek, "A recognition algorithm for deterministic CFL's optimal in time and space," *21st IEEE Symposium on Foundations of Computer Science*, pp. 411-420, (1980).
- Cook, S. A., "Deterministic CFL's are accepted simultaneously in polynomial time and log squared space," *Proceedings of the 11th ACM Symposium on Theory of Computing*, pp. 338-345, (1979).
- Cook, S.A., "Towards a complexity theory of synchronous parallel computation," presented at *Internationales Symposium über Logik und Algorithmik zu Ehren von Professor Hori Specker*, Zurich, Switzerland, February 1980.
- Fortune, S. and J. Wyllie, "Parallelism in random access machines," *Proceedings of the 10th ACM Symposium on Theory of Computation*, pp. 114-118, (1978).
- Hopcroft, J. E. and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, (1979).
- Knuth, D. E., "On the translation of languages from left to right," *Information and Control*, 8:6, pp. 607-639, (1965).
- Lewis, P. M., R. E. Stearns, and J. Hartmanis, "Memory bounds for recognition of context-free and context-sensitive languages," *IEEE Conference Record on Switching Theory and Logical Design*, pp. 191-202, (1965).
- Ruzzo, W. L., "Tree-size bounded Alternation," *Journal of Computer and System Sciences* 21, pp. 218-235, (1980).
- Ruzzo, W. L., private communication, (1982).
- Stockmeyer, L. and U. Vishkin, "Simulation of parallel random access mechanisms by circuits," IBM Yorktown Heights, (1982).



figure 1a: $\langle x, y, z \rangle_u$ in case y occurs between x and z

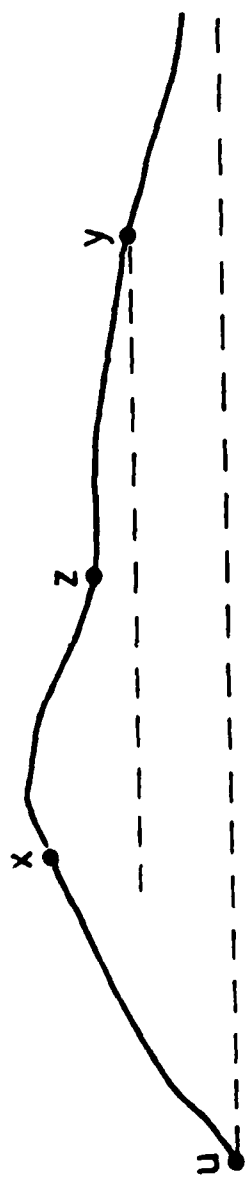


figure 1b: $\langle x, y, z \rangle_u$ in case y occurs after z

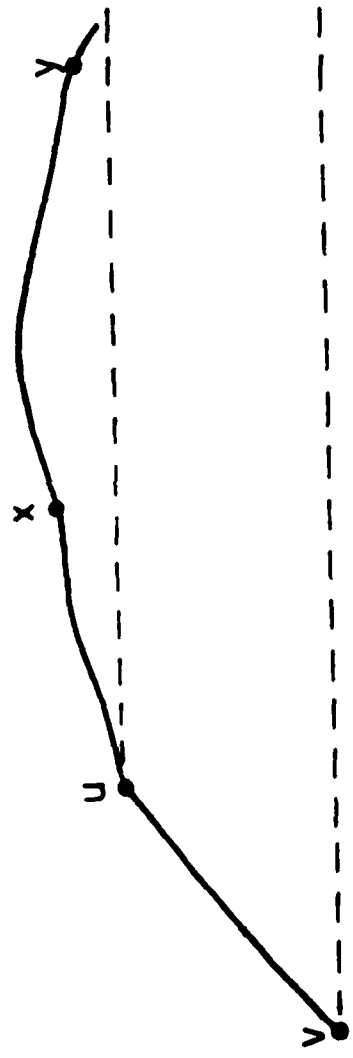


figure 2: $[x, y]_u = [x, y]_v$

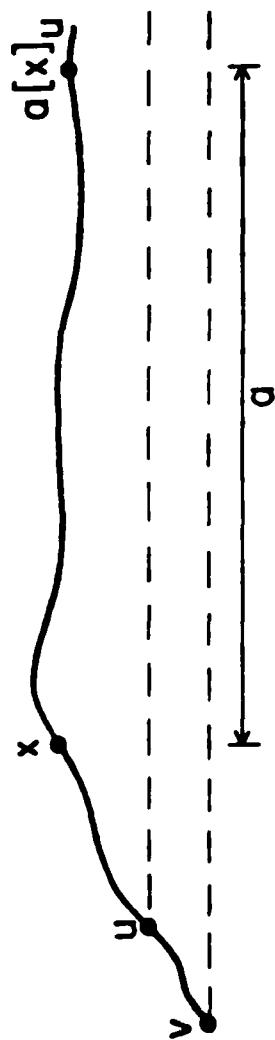


figure 3: $a[x]_u = a[x]_v$

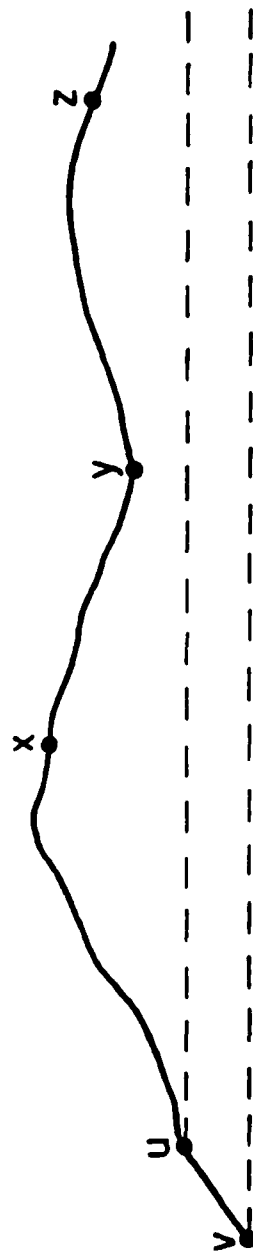


figure 4: $\langle x, y, z \rangle_u$ implies $\langle x, y, z \rangle_v$

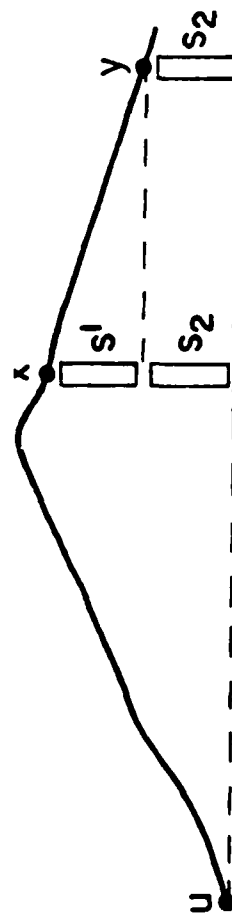


figure 5: $s_1 = s_2 \circ s'$

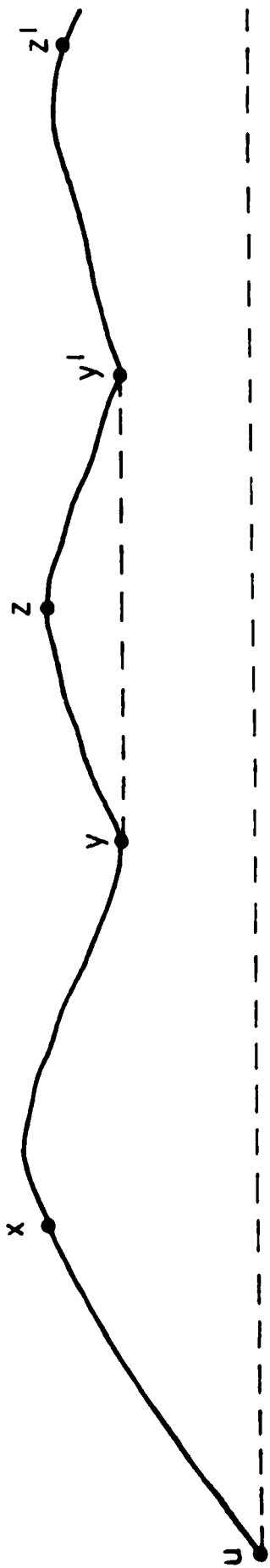


figure 6a: $\langle x, y', z' \rangle_u$ in case $h(y') \leq h(y)$

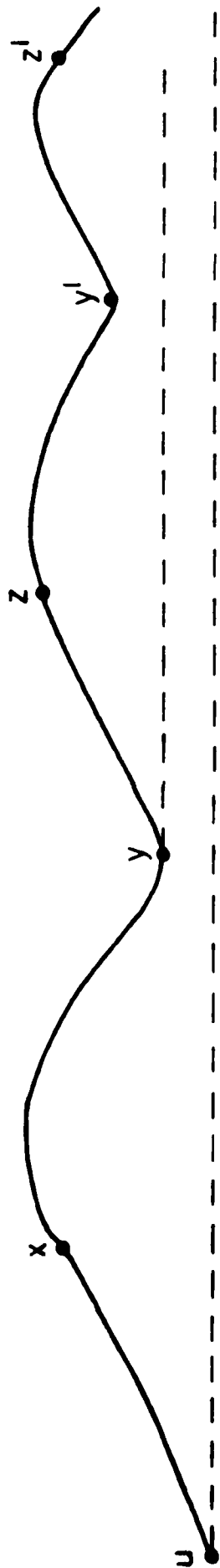


figure 6b: $\langle x, y, z' \rangle_u$ in case $h(y') > h(y)$

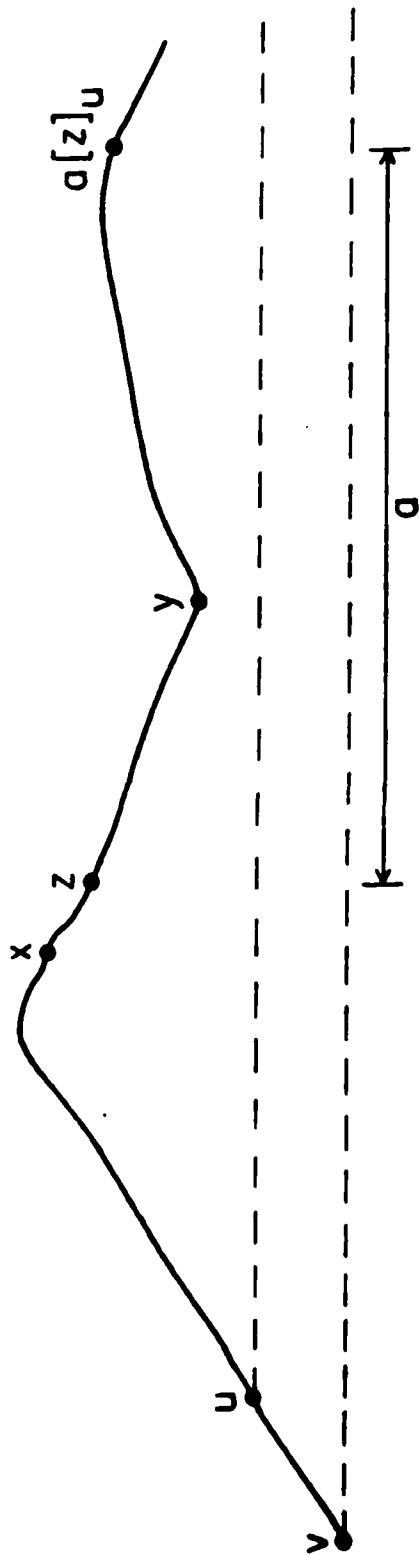


figure 7: $\langle x, y, a[z]_u \rangle_u$ implies $\langle x, y, a[z]_v \rangle_v$



figure 8a: $PREDICT_0(x,y) = z$ in case $h(z) \leq h(y)$

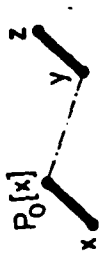


figure 8b: $PREDICT_0(x,y) = y$ in case $h(z) > h(y)$

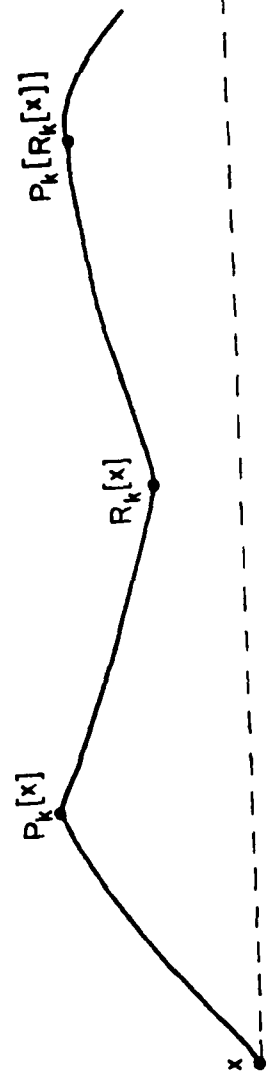


figure 9: $P_{k+1}[x] = P_k[R_k[x]]$

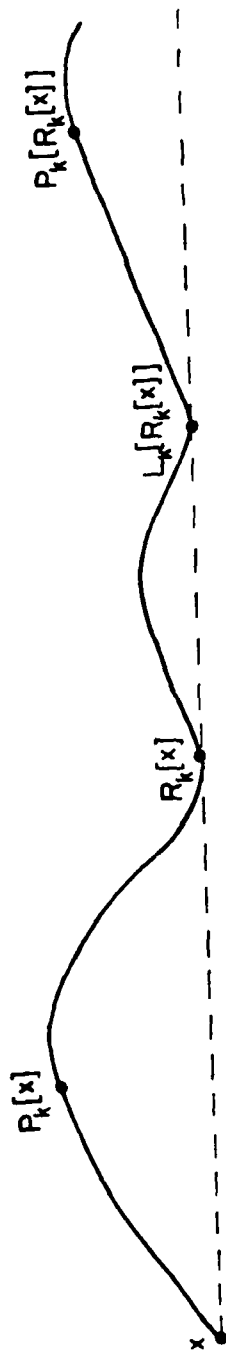


figure 10a: $L_{k+1}[x] = L_k[R_k[x]]$ in case $h(R_k[x]) = h(x)$

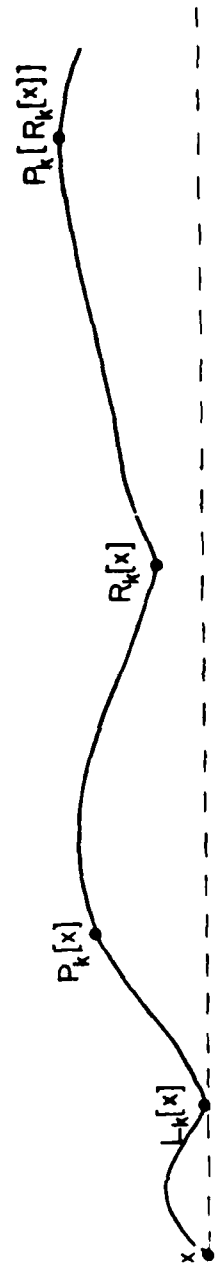


figure 10b: $L_{k+1}[x] = L_k[x]$ in case $h(R_k[x]) > h(x)$

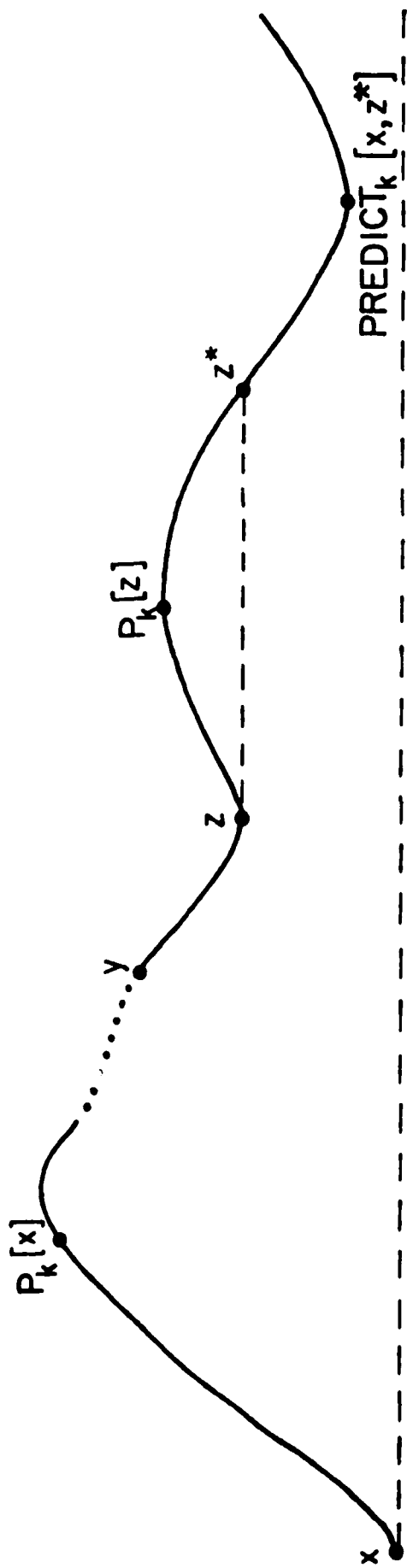


figure 11a: $\text{HOP}_{k+1}[x, y] = \text{PREDICT}_k[x, z^*]$ in case $h(z^*) = h(z)$

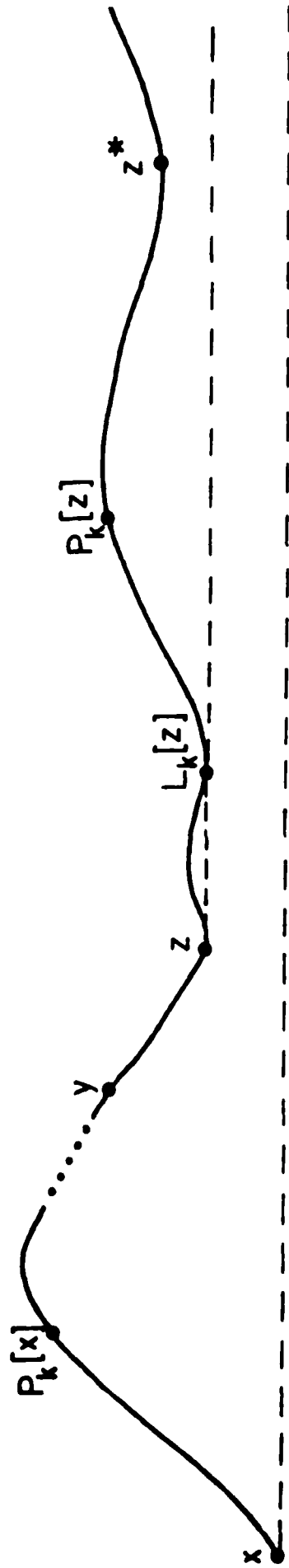


figure 11b: $\text{HOP}_{k+1}[x, y] = L_k[z]$ in case $h(z^*) > h(z)$

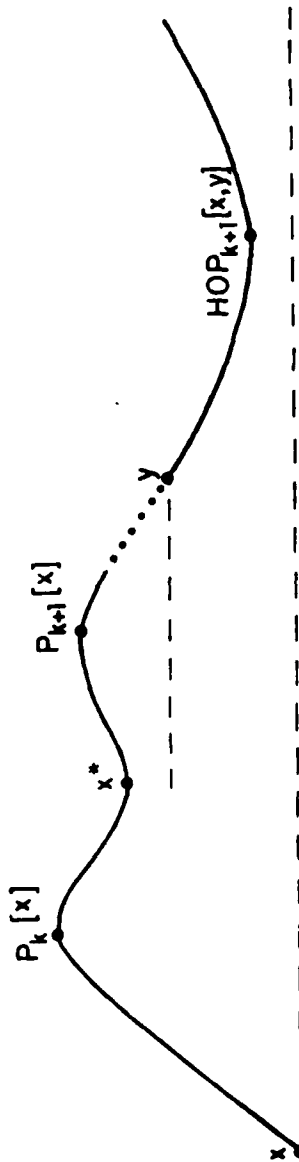


figure 12a: $PREDICT_{k+1}[x,y] = HOP_k + 1[x,y]$ in case $h(y) \leq h(x^*)$

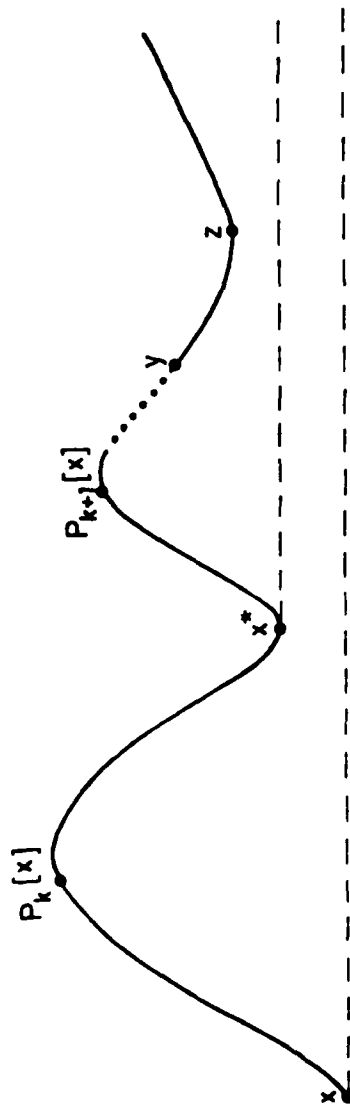


figure 12b: $PREDICT_{k+1}[x,y] = z$ in case $h(y) > h(x^*)$ and $h(z) > h(x^*)$

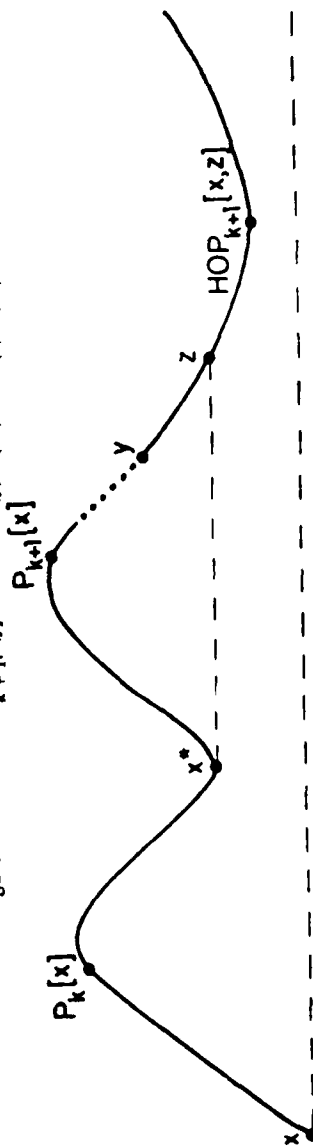


figure 12c: $PREDICT_{k+1}[x,y] = HOP_k + 1[x,z]$ in case $h(y) > h(x^*)$ and $h(z) = h(x^*)$

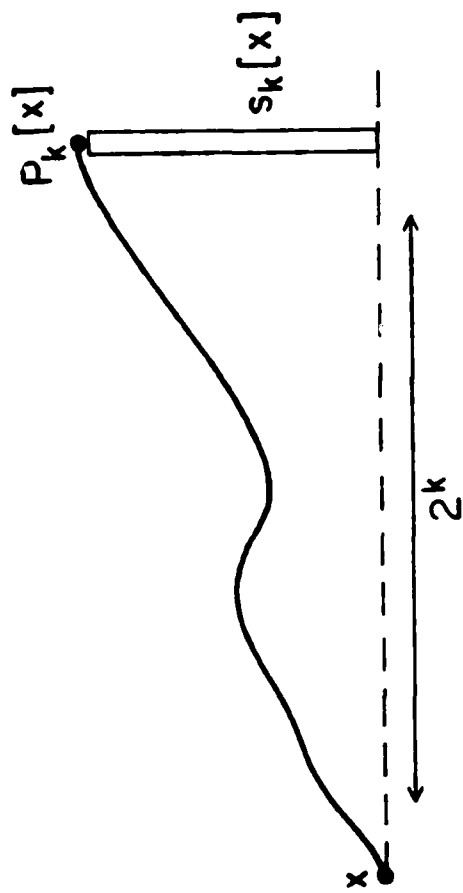


figure 13: $(\lambda, \epsilon) \vdash^a (P_k[x], s_k[x])$ for some $a \geq 2k$

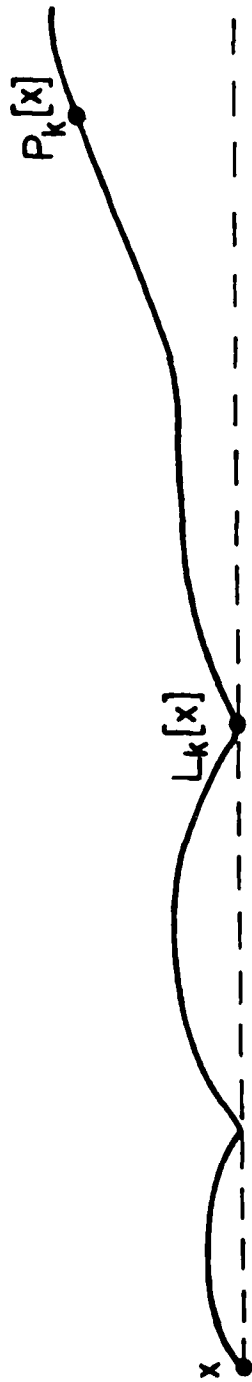


figure 14: $\langle \lambda, L_k[x], P_k[x] \rangle_x$ and $h(L_k[x]) = h(x)$

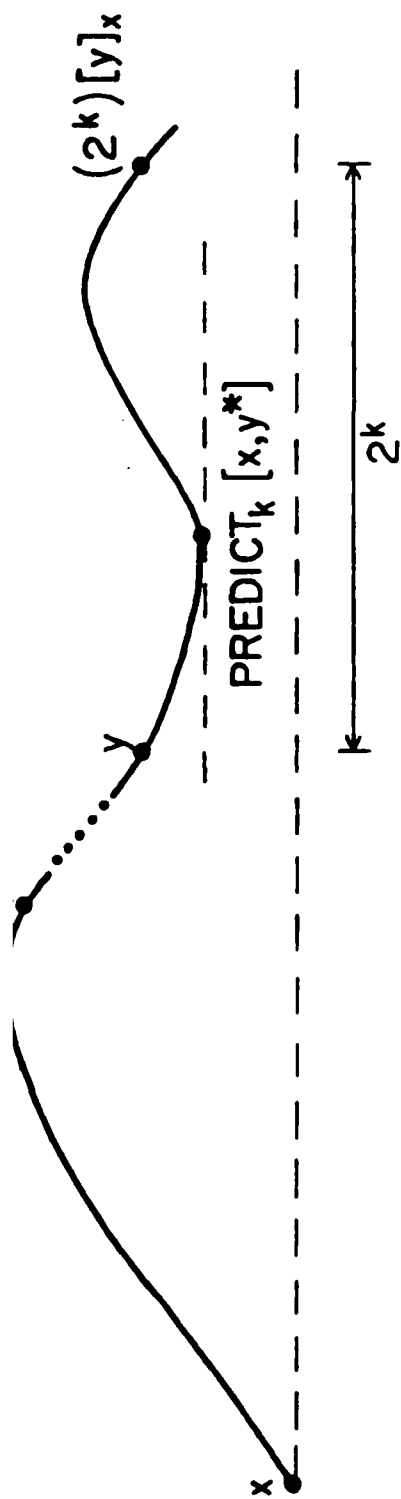


figure 15: $\langle y, \text{PREDICT}_k[x, y], (2^k)[y]_x \rangle_x$

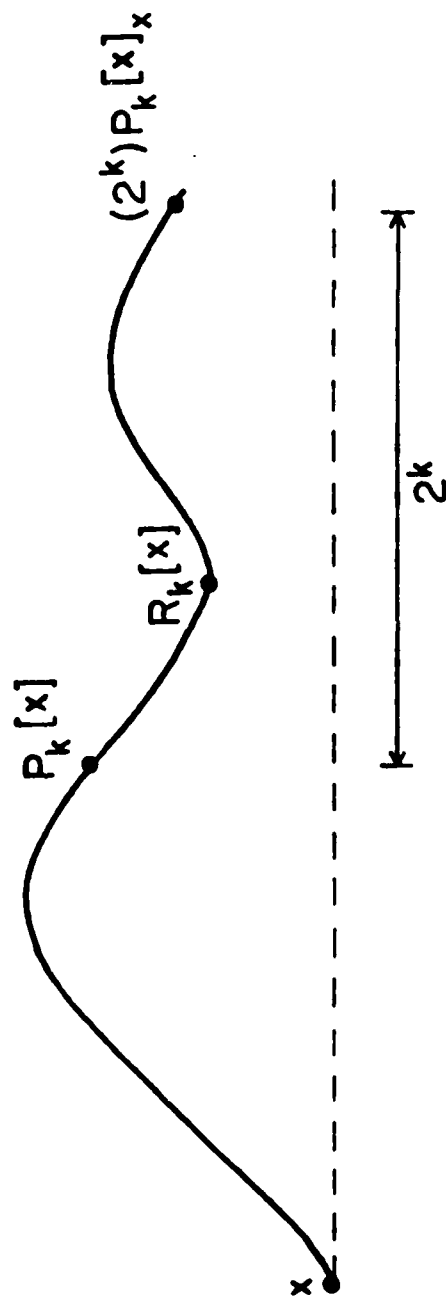


figure 16: $\langle P_k[x], R_k[x], (2^k)[P_k[x]]_x \rangle_x$

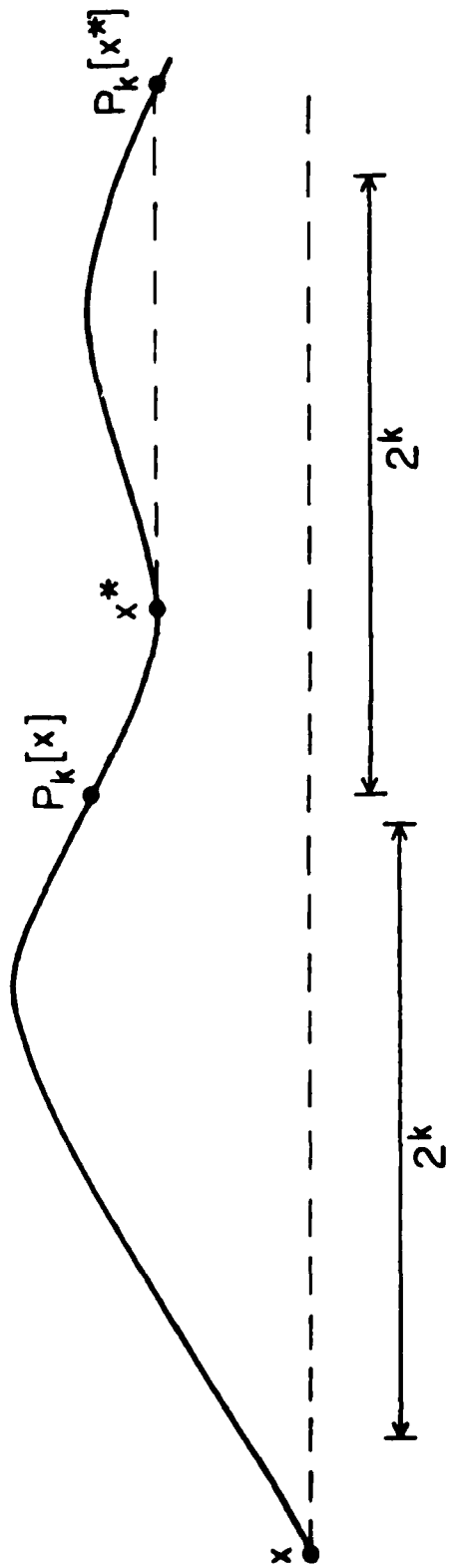


figure 17: $P_k[x^*] \in (x^*, (2^k)P_k[x])_{x^*}$ because $h(P_k[x^*]) = h(x^*)$

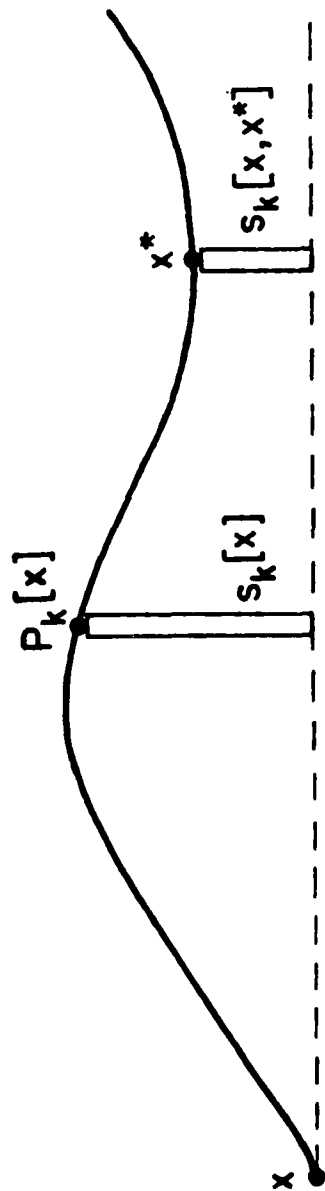


figure 18: $(x, \epsilon) \vdash^* (x^*, S_k[x, x^*])$

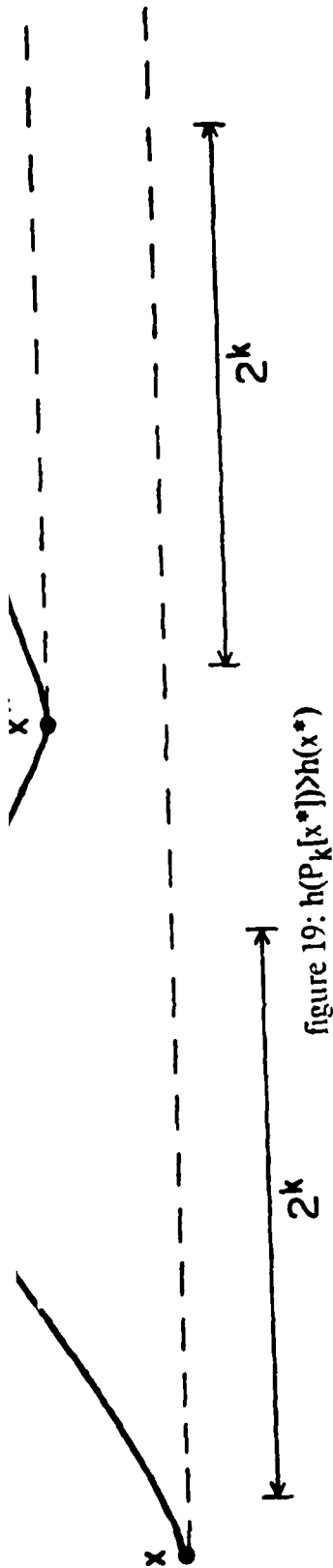


figure 19: $h(P_k[x^*]) > h(x^*)$

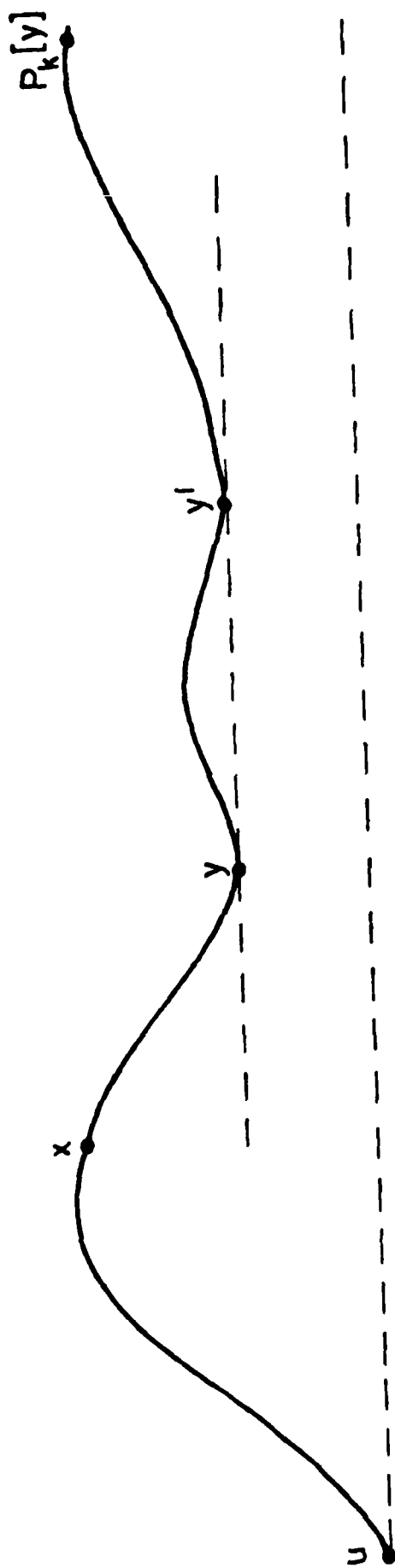


figure 20: $\langle x, y, P_k[y] \rangle_u$

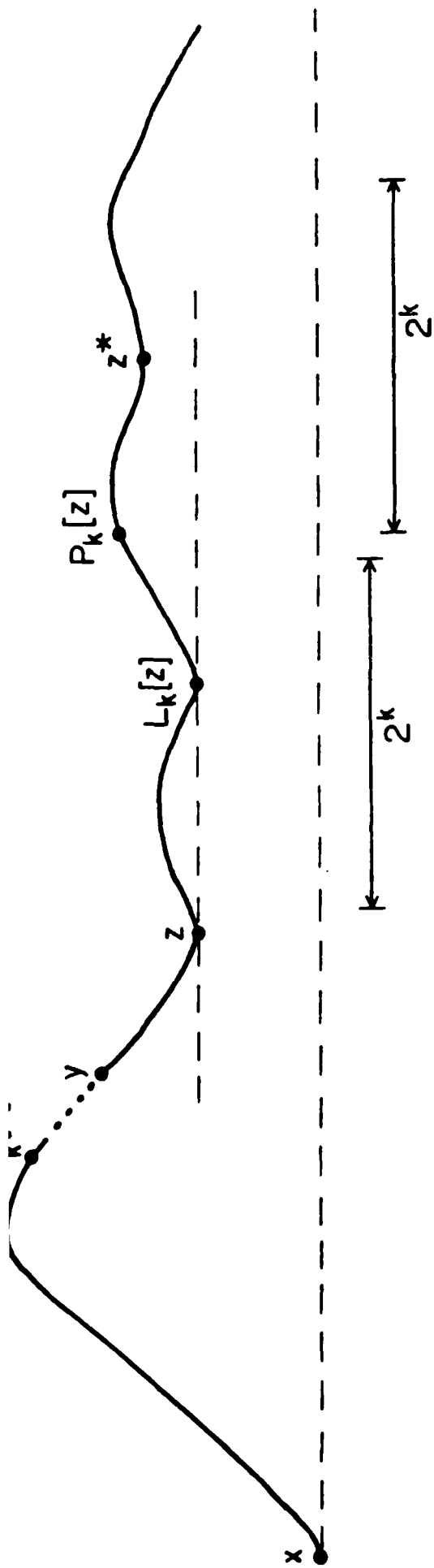


figure 21a: $HOP_{k+1}[x,y] = L_k[z]$ in case $h(z^*) > h(z)$

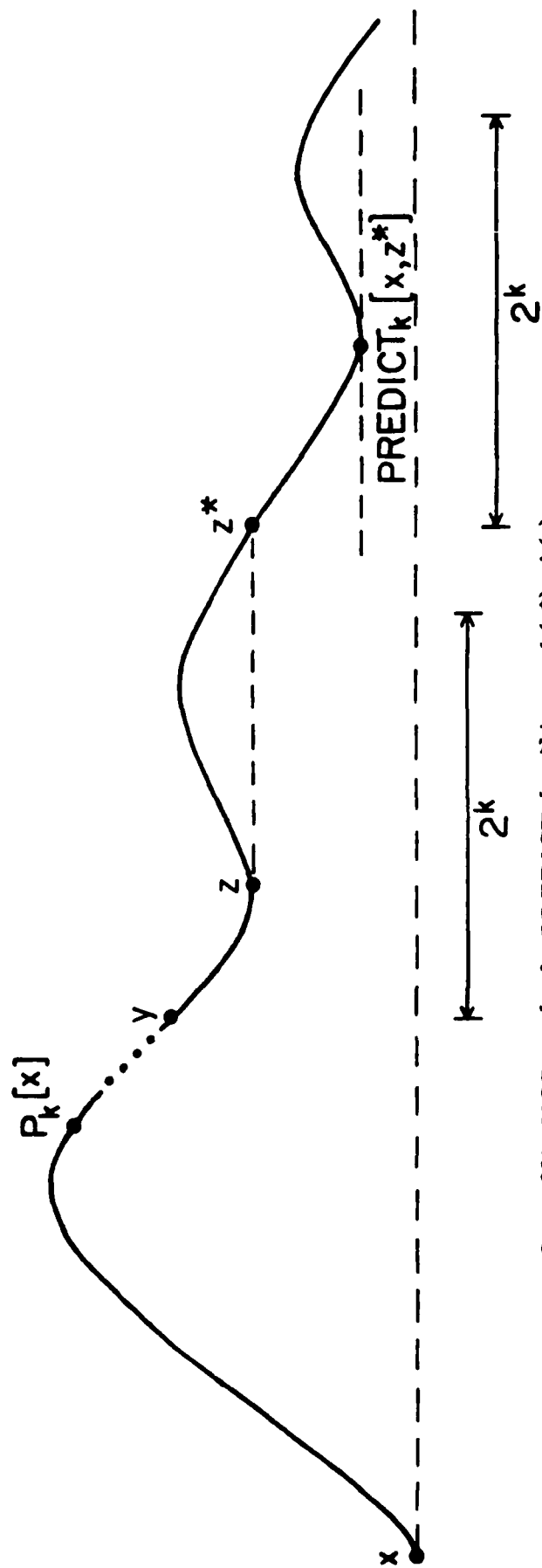


figure 21b: $HOP_{k+1}[x,y] = PREDICT_k[x,z^*]$ in case $h(z^*) = h(z)$

END

FILMED

5-85

DTIC